Overview

Jenkins is a very popular Java-based open source continuous integration (CI) server that allows teams to continuously build applications across platforms. Azure Pipeline includes the ability to build any application on any platform including Windows, Linux and Mac. However, it also integrates well with Jenkins for teams who already use or prefer to use Jenkins for CI.

There are two ways to integrate Jenkins with Azure Pipelines:

- One way is to **run CI jobs in Jenkins** separately. This involves configuration of a CI pipeline in Jenkins and a web hook in Azure DevOps that invokes the CI process when source code is pushed to a repository or a branch.
- The alternate way is to **wrap a Jenkins CI job inside an Azure pipeline**. In this approach, a build definition will be configured in Azure Pipelines to use the **Jenkins** tasks to invoke a CI job in Jenkins, download and publish the artifacts produced by Jenkins.

An Azure CD pipeline can be configured to pick these build artifacts, irrespective of the approach, for deployment. While there are pros and cons with both the approaches, the latter approach has multiple benefits:

1. End-to-end traceability from work item to source code to build and release
2. Triggering of a Continuous Deployment (CD) when the build is completed successfully
3. Execution of the build as part of the branching strategy

What's covered in this lab

This lab covers both the approaches and the following tasks will be performed

- Provision Jenkins on Azure VM using the Jenkins template available on the Azure Marketplace
- Configure Jenkins to work with Maven and Azure DevOps
- Create a build job in Jenkins
- Configure Azure Pipeline to integrate with Jenkins
- Configure a CD pipeline in Azure Pipelines to deploy the build artifacts

Before you begin

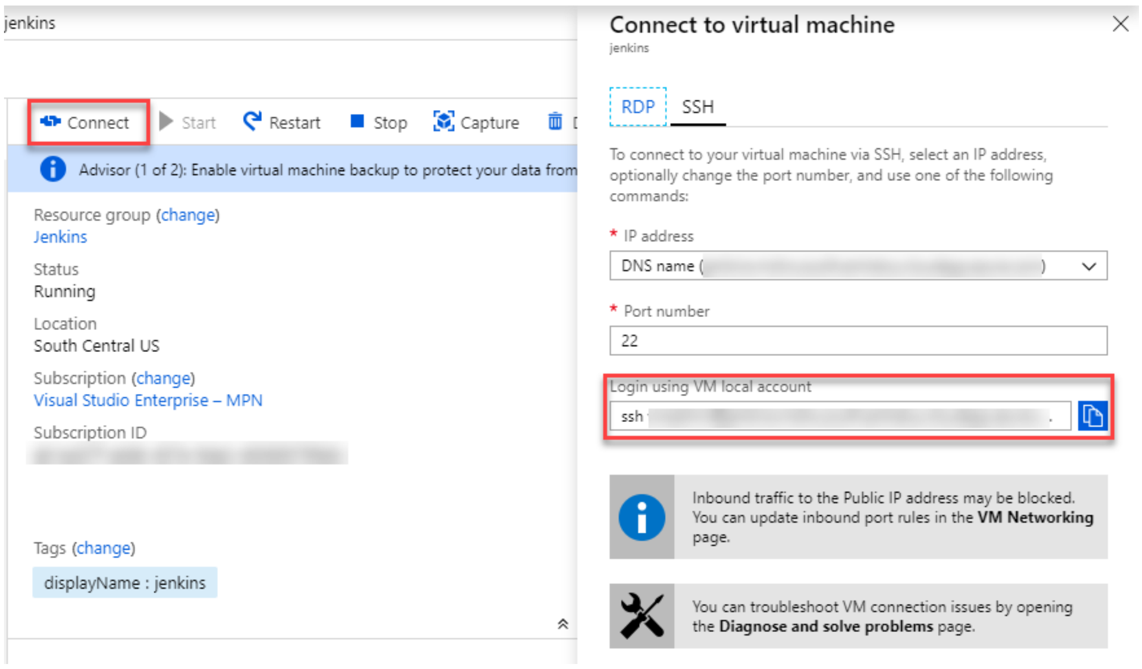
1. **Microsoft Azure Account**: You will need a valid and active Azure account for the Azure labs. If you do not have one, you can sign up for a free trial
2. Putty a free SSH and Telnet client
3. Set up your Azure DevOps project using the **MyShuttle** template in the Azure DevOps Demo Generator. We will use a Java web app that connects to a MySQL backend.

Setting up the Jenkins VM

1. To configure Jenkins, the Jenkins VM image available on the Azure MarketPlace will be used. This will install the latest stable Jenkins version on an Ubuntu Linux VM along with the tools and plugins configured to work with Azure. Click the **Deploy to Azure** button below to get started.

   Deploy to Azure

2. Once the Jenkins VM is provisioned, select the **Connect** button and make a note of the `username` and the `ip address`. This information will be required to connect to the Jenkins VM from ***Putty***



jenkins

Connect | Start | Restart | Stop | Capture

Advisor (1 of 2): Enable virtual machine backup to protect your data from

Resource group (change)
Jenkins

Status
Running

Location
South Central US

Subscription (change)
Visual Studio Enterprise – MPN

Subscription ID
[redacted]

Tags (change)
displayName : jenkins

Connect to virtual machine
jenkins

RDP | SSH

To connect to your virtual machine via SSH, select an IP address, optionally change the port number, and use one of the following commands:

* IP address
DNS name ([redacted])

* Port number
22

Login using VM local account
ssh [redacted]

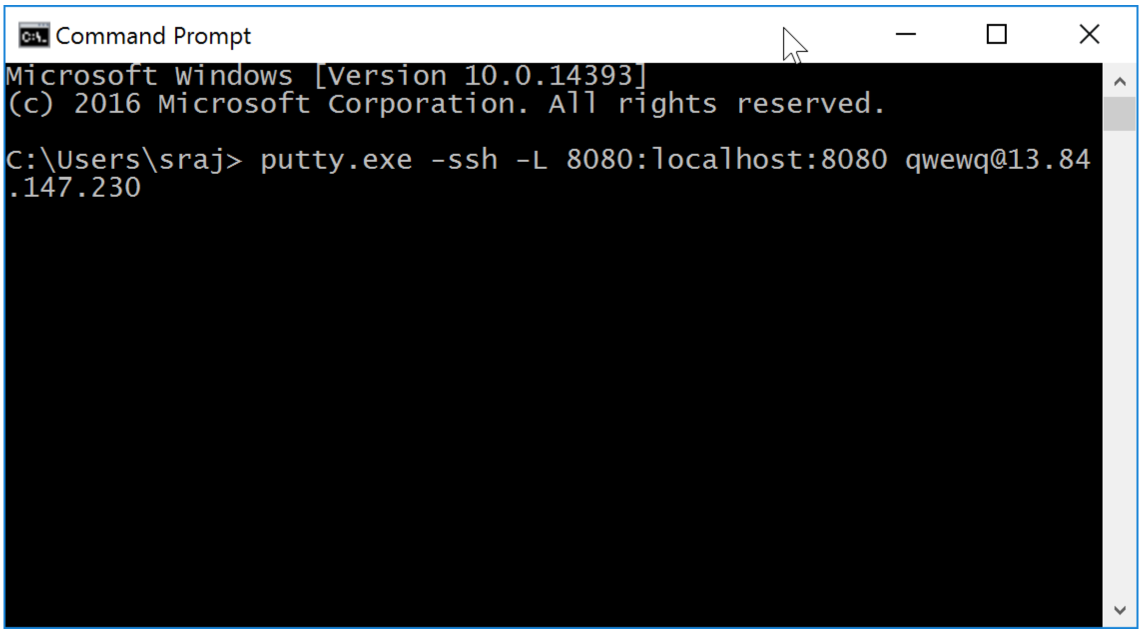
Inbound traffic to the Public IP address may be blocked. You can update inbound port rules in the **VM Networking** page.

You can troubleshoot VM connection issues by opening the **Diagnose and solve problems** page.

Note: Jenkins, by default, listens on port 8080 using HTTP. To configure a secure HTTPS connection, an SSL certificate will be required. If HTTPS communication is not being configured, the best way to ensure the sign-in credentials are not leaked due to a “Man-in-the-middle” attack is by logging-in using the SSH tunneling. An SSH tunnel is an encrypted tunnel created through an SSH protocol connection, that can be used to transfer unencrypted traffic over an unsecured network.

3. To initiate an SSH tunnel, the following command needs to be run from a Command Prompt. An SSH tunnel creates a secure connection between your host and remote computer through which services can be relayed. If this command is successful, you should be able to access the remote Jenkins on port 8080 on your local machine.

4.
```
putty.exe -ssh -L 8080:localhost:8080 <username>@<ip address>
```



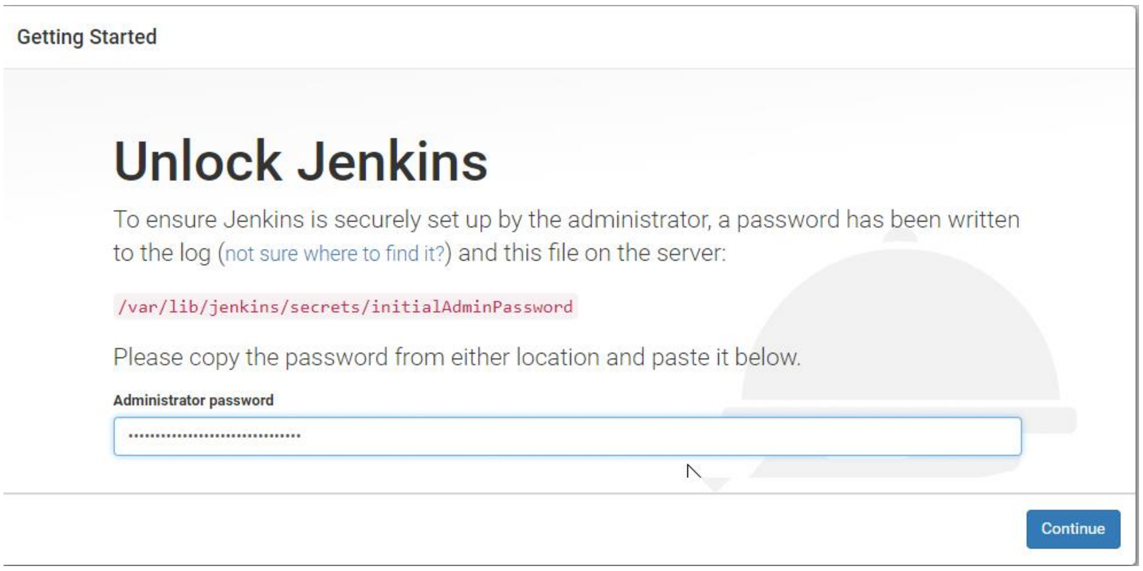
```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\sraj> putty.exe -ssh -L 8080:localhost:8080 qwewq@13.84
.147.230
```

Note: To run the above command, either the Putty.exe needs to be placed in the path selected in the Command Prompt or the full path of the Putty.exe need to be provided in the command.

5. Log in with the user name and password that you provided while provisioning the Jenkins VM.
6. Once the connection is successful, open a browser on the host machine and navigate to the URL http://localhost:8080. The **Getting Started** page for Jenkins will be displayed.
7. For security reasons, Jenkins generates an initial password and save it in a file on the server. This password will need to be retrieved and provided to unlock Jenkins. Return to the **Putty** terminal and type the following command to open the password file and copy the password. You can double click on the password text and use **CTRL+C** to copy the text and place it in the clipboard. Press the **Esc** button and then type **:q!** at the prompt to exit the vi editor without saving the file.

```
sudo vi /var/lib/jenkins/secrets/initialAdminPassword
```

8. Return to the browser, paste the copied text in the Administrator password text box and click on the **Continue** button.


Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

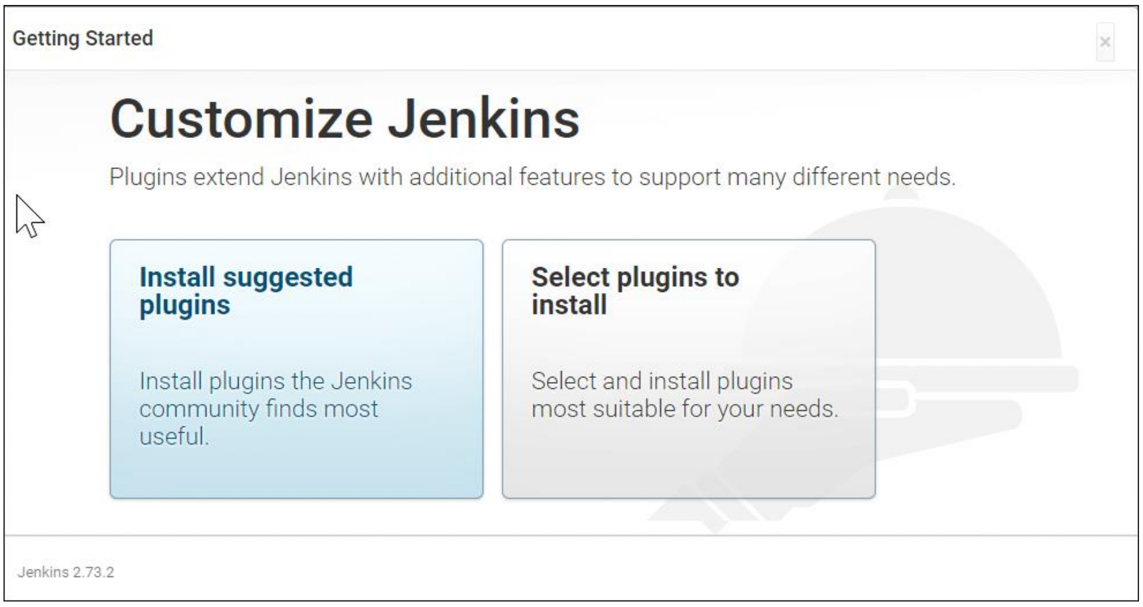
`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

••••••••••••••••••••••••••••••••

9. Jenkins has a vast ecosystem with a strong and active open source community users contributing hundreds of useful plugins. While configuring Jenkins, you can choose between installing the most commonly used plugins or go for specific selected plugins. Select **Install suggested plugins** to initiate the configuration with default plugins. We will install other plugins such as Maven, Azure DevOps manually later.


Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

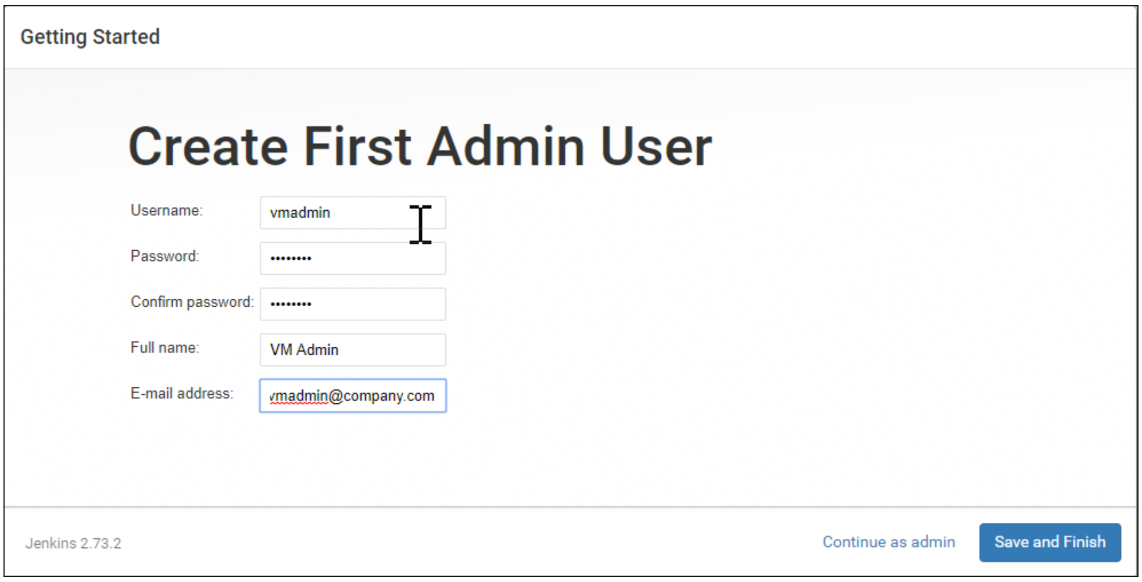
Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.73.2

10. The final step is to create a new `Admin` user. Provide *User name*, *Password*, *Full name* and *Email address* for the user. Select **Save and Finish** when you are done.


Getting Started

Create First Admin User

Username: vmadmin

Password: ••••••••

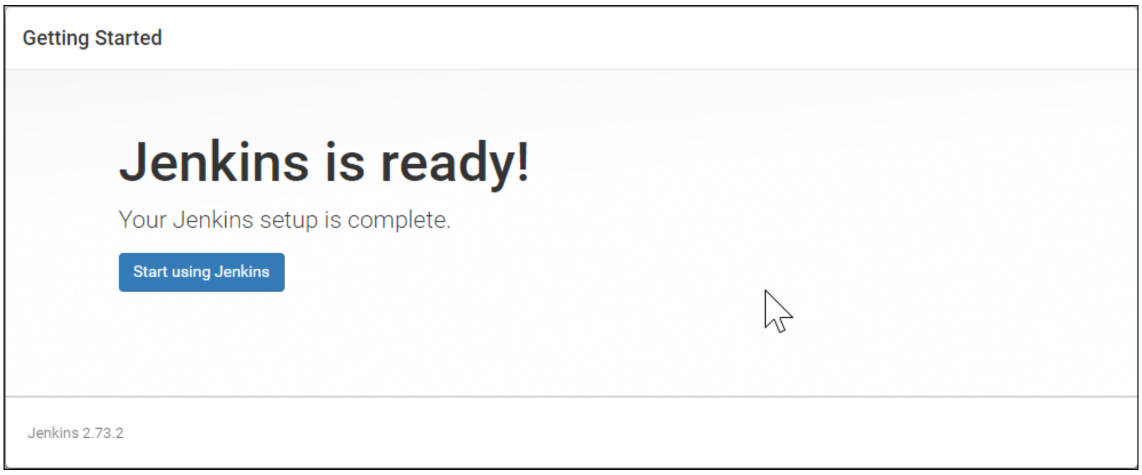
Confirm password: ••••••••

Full name: VM Admin

E-mail address: vmadmin@company.com

Jenkins 2.73.2

11. Jenkins is now ready for use. Select **Start using Jenkins** to start using it.


Getting Started

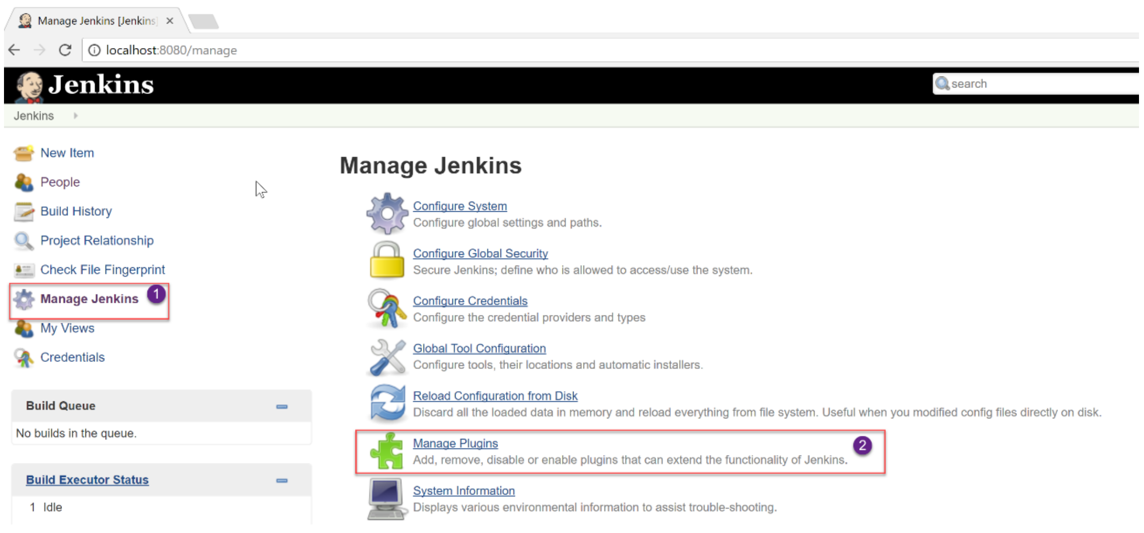
Jenkins is ready!

Your Jenkins setup is complete.

Jenkins 2.73.2

Installing and Configuring Plugins

1. We will now install the Maven and VSTS (yet to be renamed Azure DevOps!) plugins that we require for this lab. Click **Manage Jenkins** on the Jenkins home page and select **Manage Plugins**. Select the **Available** tab and search for `team services`



Manage Jenkins

Configure System
Configure global settings and paths.

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

Configure Credentials
Configure the credential providers and types

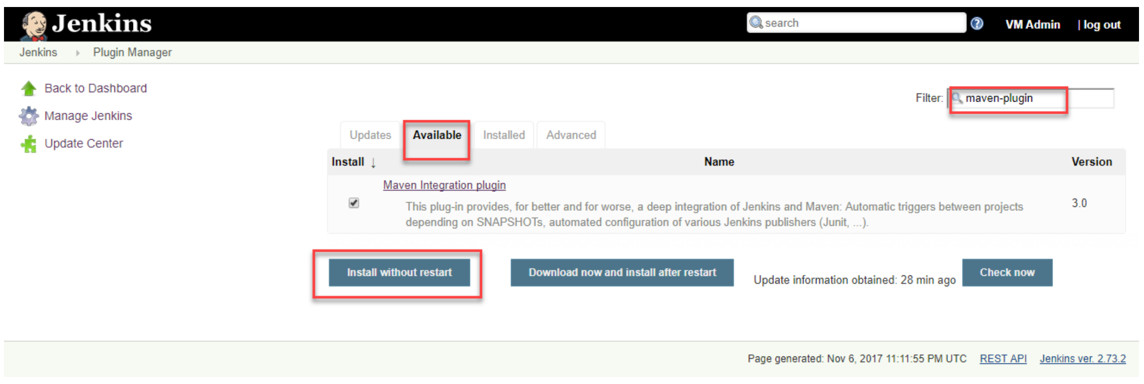
Global Tool Configuration
Configure tools, their locations and automatic installers.

Reload Configuration from Disk
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

System Information
Displays various environmental information to assist trouble-shooting.

2. Select **VS Team Services Continuous Deployment** plugin and select **Install without restart**
3. Select **Manage Plugins**, select the **Available** tab and search for `maven-plugin`
4. Select the **Maven Integration Plugin** and select the **Install without restart** button to install the plugin.

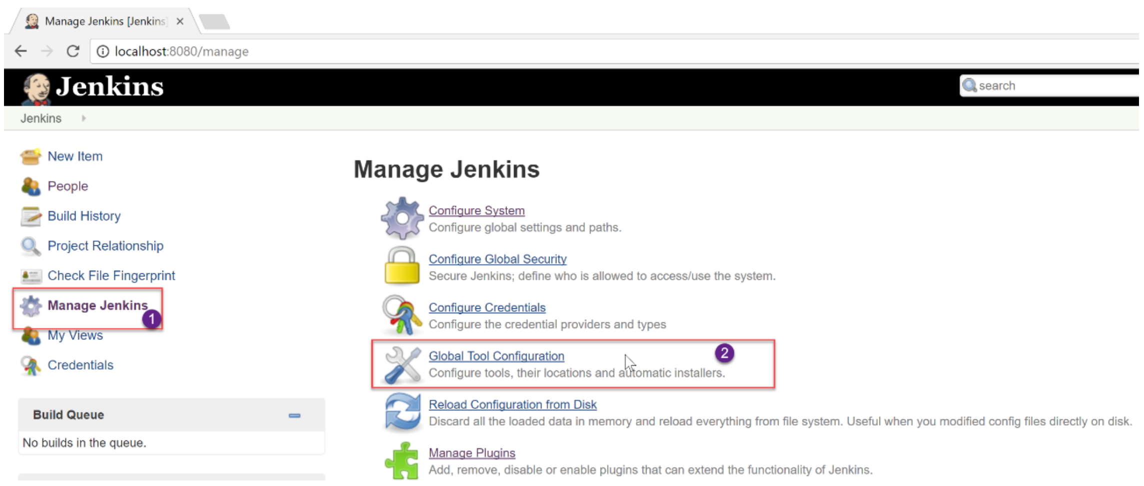


Install ↓	Name	Version
✓	Maven Integration plugin This plug-in provides, for better and for worse, a deep integration of Jenkins and Maven: Automatic triggers between projects depending on SNAPSHOTs, automated configuration of various Jenkins publishers (Junit, ...).	3.0

Update information obtained: 28 min ago

Page generated: Nov 6, 2017 11:11:55 PM UTC REST API Jenkins ver. 2.73.2

5. Once the plugin is installed, go back to **Manage Jenkins** and select the **Global Tool Configuration** option.



Manage Jenkins

Configure System
Configure global settings and paths.

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

Configure Credentials
Configure the credential providers and types

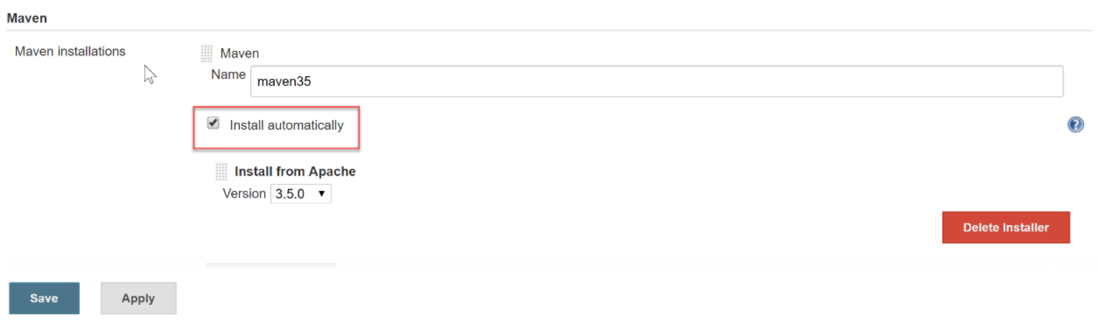
Global Tool Configuration
Configure tools, their locations and automatic installers.

Reload Configuration from Disk
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Note: Jenkins provides great out-of-the-box support for Maven. Since Maven is not yet installed, it can be manually installed by extracting the `tar` file located in a shared folder. Alternatively, when the **Install automatically** option is selected in the **Global Tool Configuration** screen, Jenkins will download and install Maven from the Apache website when a build job requires it.

6. To install Maven, select the **Install automatically** option and select the **Apply** button. The latest version of Maven at the time of writing this lab is 3.5.4



Maven

Maven installations

Maven

Name: maven35

☑ Install automatically

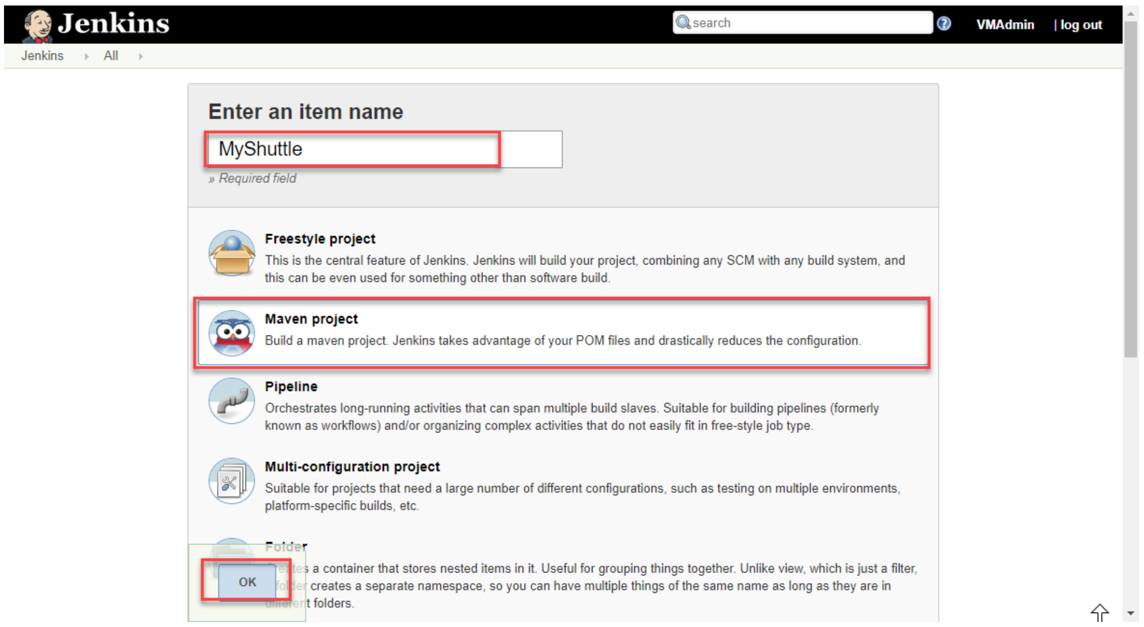
Install from Apache

Version: 3.5.0

7. Select the **Back to Dashboard** button to return to the home page. We are done with the setup. Let's go and create a new CI job.

Creating a new build job in Jenkins

1. From the Jenkins home page, click the**New Item** link. Provide a name for the build definition, select **Maven project** and click **OK**.



Enter an item name

MyShuttle

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

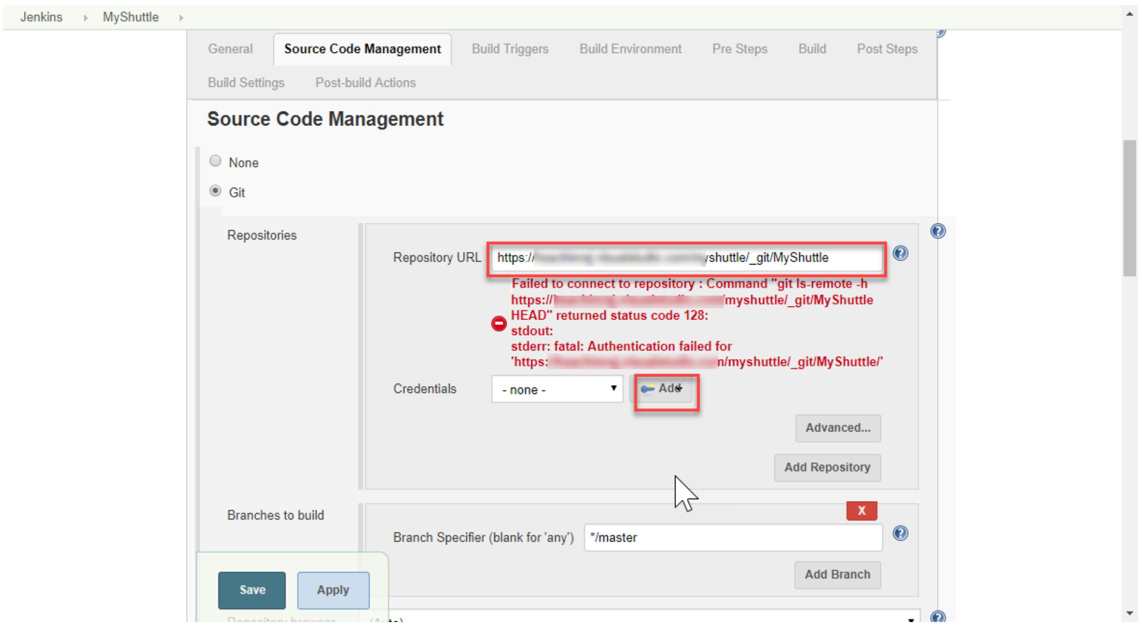
Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

2. Now scroll down to the **Source code Management** section. Select **Git** and provide the clone URL of the Azure DevOps Git repo in the format `http://dev.azure.com/{your org name}/{team project name}/_git/MyShuttle`. If you do not see **Git** under Source code management (not a usual thing), you will need to install/enable the Git plugin.



Source Code Management

None [✓ Git]

Repositories

Repository URL: https://[redacted]shuttle/_git/MyShuttle

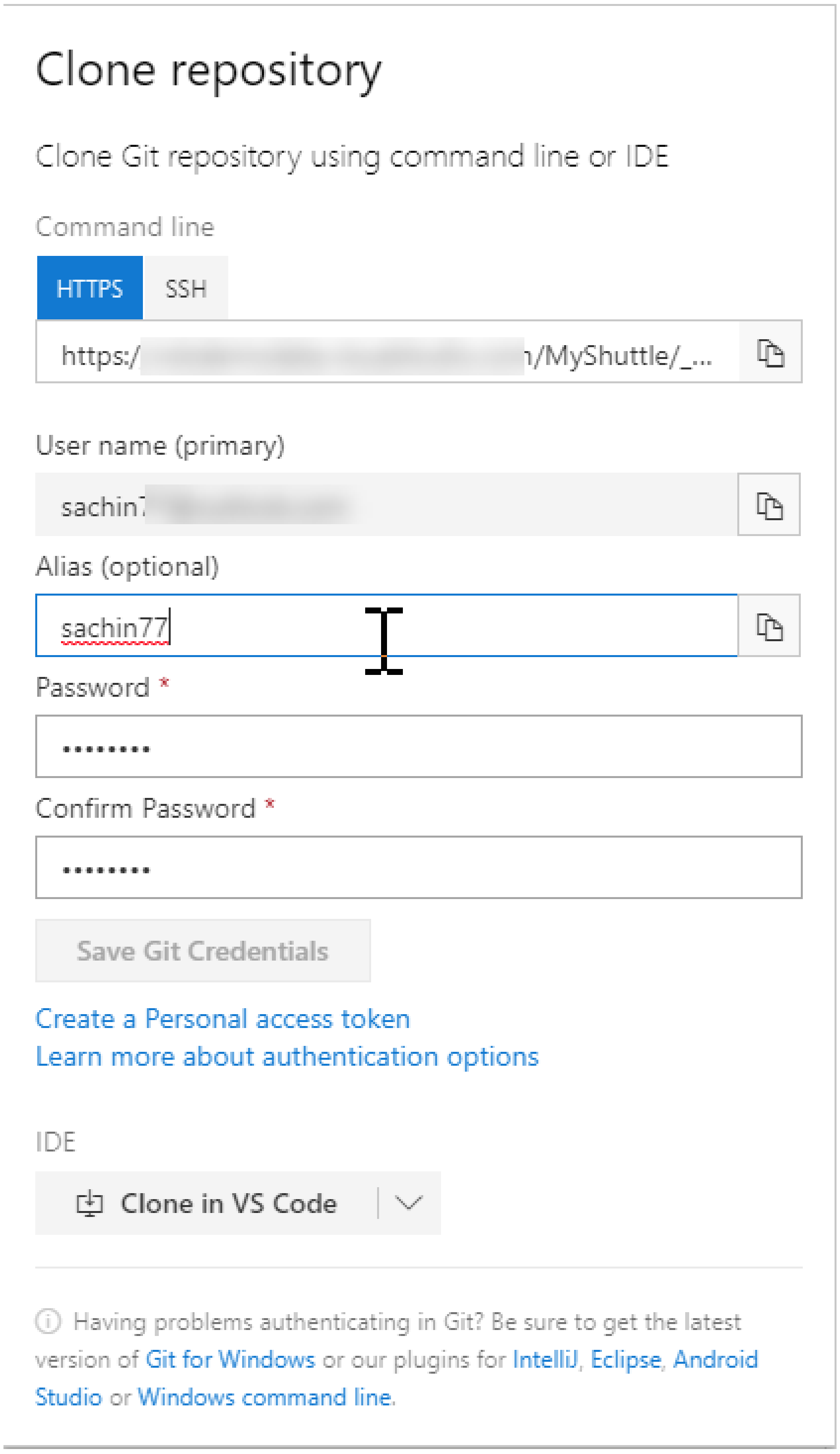
Failed to connect to repository : Command "git ls-remote -h https://[redacted]myshuttle/_git/MyShuttle HEAD" returned status code 128:
stdout:
stderr: fatal: Authentication failed for 'https:[redacted]n/myshuttle/_git/MyShuttle/'

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */master

3. Your Azure repo is very likely to be private. Unless you have a public repo, you should provide the credentials to access the repository. If you do not have one or don't remember the credentials, go to your Azure Repos and select the **Clone** option. Select **Generate Credentials** and enter a `User name` and `Password`. Click **Save Git Credentials** to save.



Clone repository

Clone Git repository using command line or IDE

Command line

HTTPS SSH

https:/[redacted]/MyShuttle/_...

User name (primary)

sachin7[redacted]

Alias (optional)

sachin77

Password *

••••••••

Confirm Password *

••••••••

Save Git Credentials

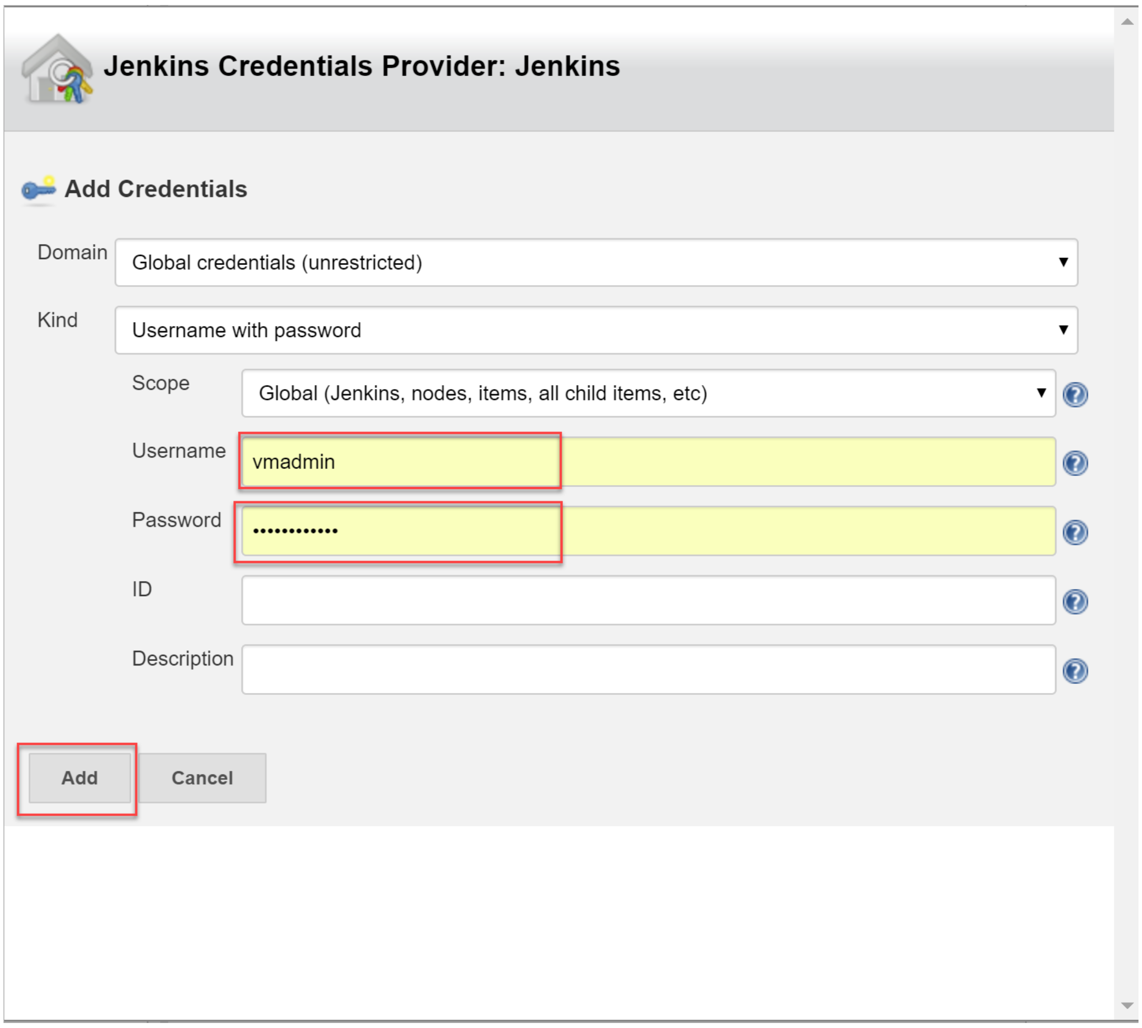
Create a Personal access token
Learn more about authentication options

IDE

Clone in VS Code

ⓘ Having problems authenticating in Git? Be sure to get the latest version of Git for Windows or our plugins for IntelliJ, Eclipse, Android Studio or Windows command line.

4. Select the **Add | Jenkins** option to add a new credential. Provide the `User name` and `Password` created earlier and click the **Add** button to close the wizard

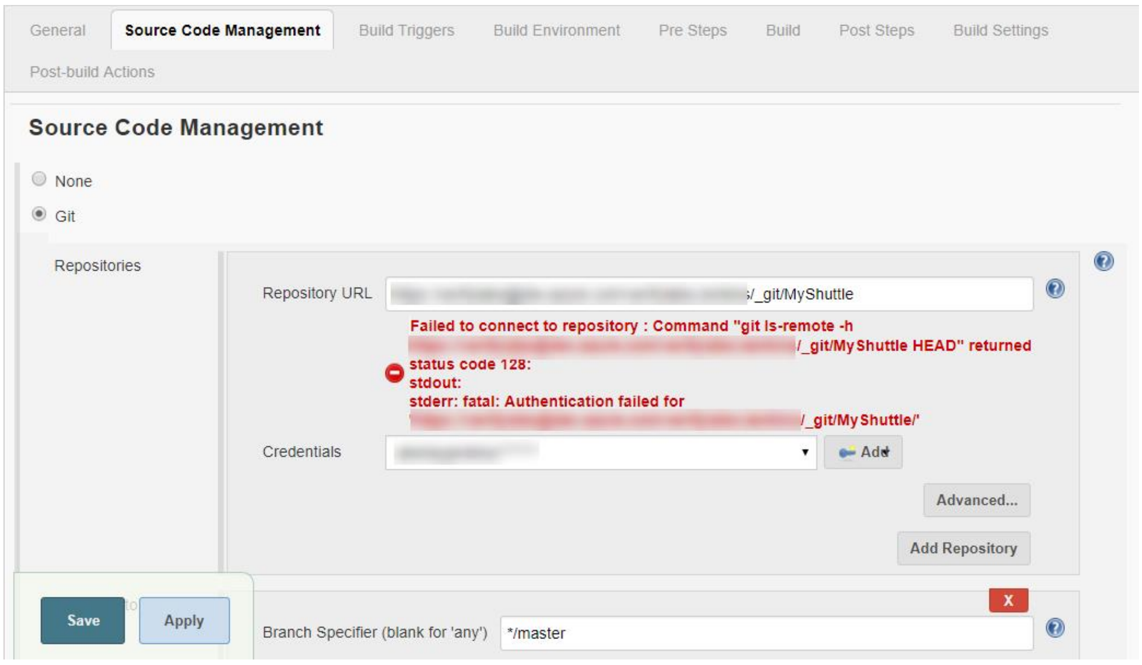


Jenkins Credentials Provider: Jenkins

Add Credentials

Domain	Global credentials (unrestricted)
Kind	Username with password
Scope	Global (Jenkins, nodes, items, all child items, etc)
Username	vmadmin
Password	••••••••••••
ID	
Description	

5. Select the credential created in the previous step from the drop-down. The error message should disappear.



Source Code Management

None

[✓ Git]

Repositories

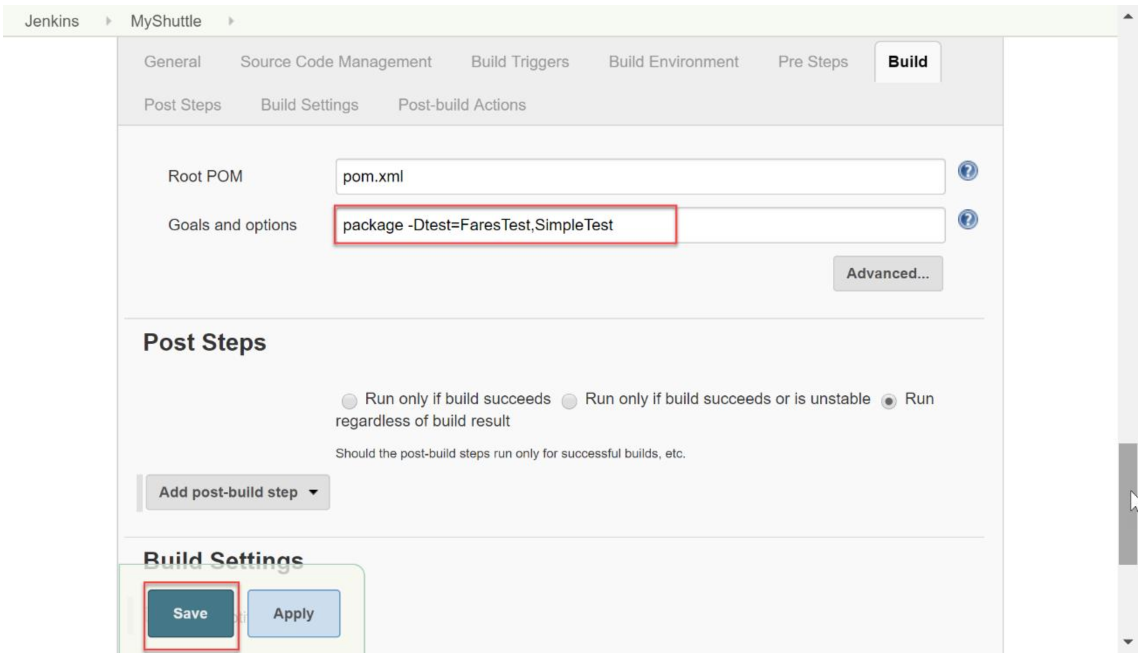
Repository URL [redacted]/_git/MyShuttle

```
Failed to connect to repository : Command "git ls-remote -h
[redacted]/_git/MyShuttle HEAD" returned
status code 128:
stdout:
stderr: fatal: Authentication failed for
[redacted]/_git/MyShuttle/'
```

Credentials [redacted]

Branch Specifier (blank for 'any') */master

6. The source code for this application has both unit tests and UI tests. We are not ready to run the UI test at this point. So, we will specify to run only the unit tests. Scroll down to the **Build** section and provide the text `package -Dtest=FaresTest,SimpleTest` in the **Goals and options** field.



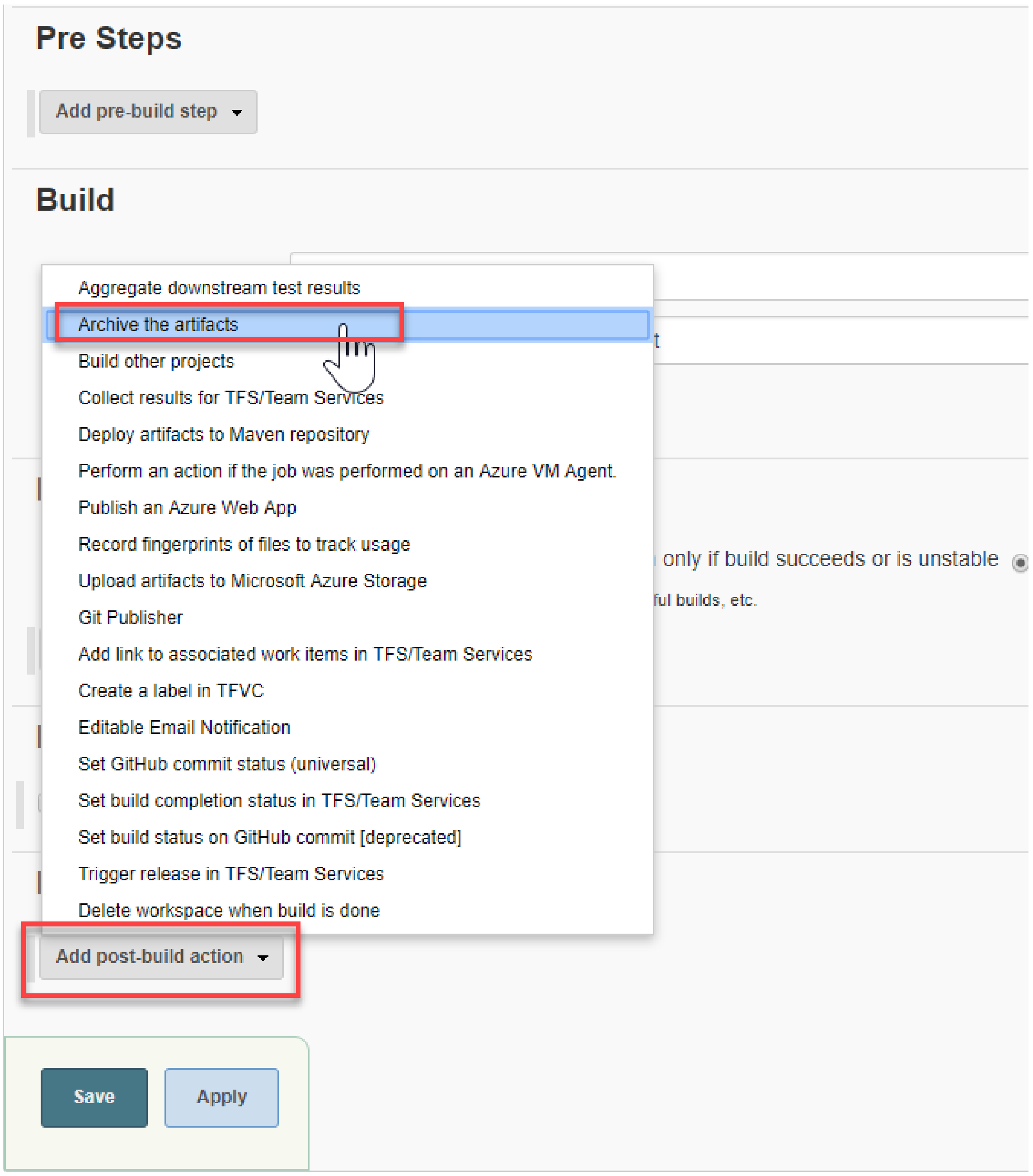
Root POM: pom.xml

Goals and options: package -Dtest=FaresTest,SimpleTest

Run only if build succeeds Run only if build succeeds or is unstable [✓ Run regardless of build result]

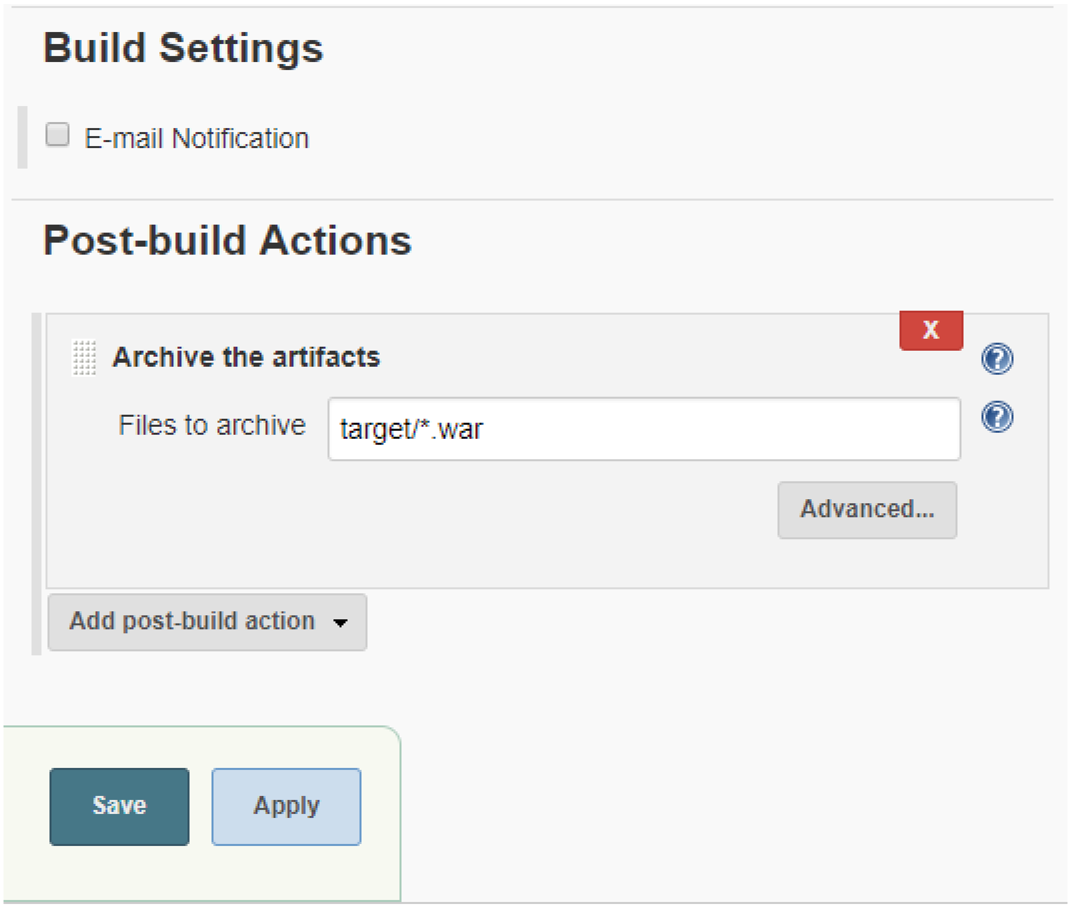
Should the post-build steps run only for successful builds, etc.

7. Once the build is complete, you can specify what action you want to take. For instance, you can archive the build artifacts, trigger an Azure CD pipeline, deploy directly to Azure App Service, etc., We will choose the **Archive the artifacts** option in the **Post-build Actions**.


Pre Steps
Add pre-build step
Build
Aggregate downstream test results
Archive the artifacts
Build other projects
Collect results for TFS/Team Services
Deploy artifacts to Maven repository
Perform an action if the job was performed on an Azure VM Agent.
Publish an Azure Web App
Record fingerprints of files to track usage
Upload artifacts to Microsoft Azure Storage
Git Publisher
Add link to associated work items in TFS/Team Services
Create a label in TFVC
Editable Email Notification
Set GitHub commit status (universal)
Set build completion status in TFS/Team Services
Set build status on GitHub commit [deprecated]
Trigger release in TFS/Team Services
Delete workspace when build is done
only if build succeeds or is unstable
ful builds, etc.
Add post-build action
Save
Apply

Note: Note there is also **Post-build steps** section which is very similar to the actions section. The tasks configured in the post-build steps/actions are executed after all the build steps have been executed.

8. Enter **target/*.war** in the **Files to archive** text box. Select the **Save** button to save the settings and return to the project page.



Build Settings

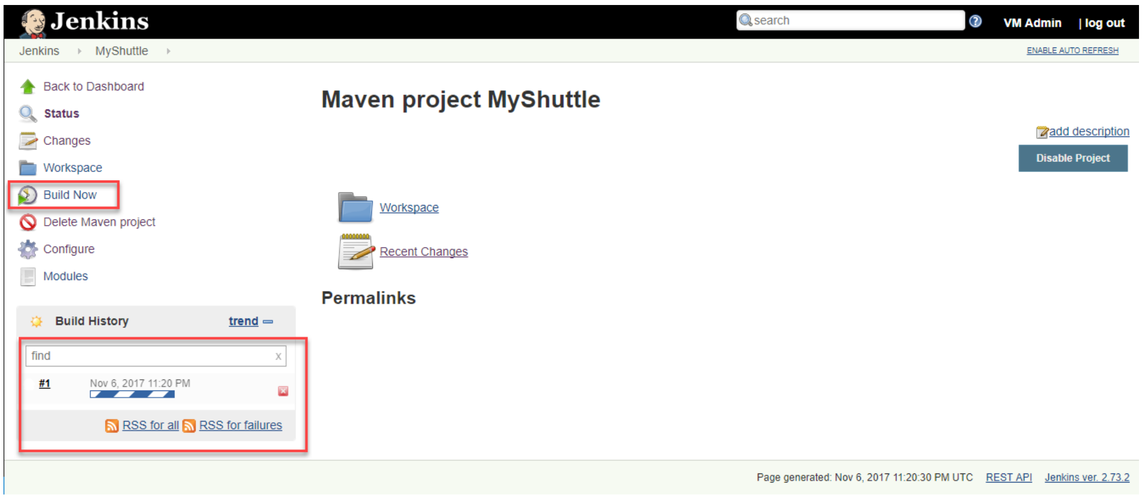
E-mail Notification

Post-build Actions

Archive the artifacts

Files to archive: target/*.war

9. The configuration is now completed, Select the **Build Now** option to initiate an Ad-hoc build. The build progress will be displayed on the left pane in the **Build History** section

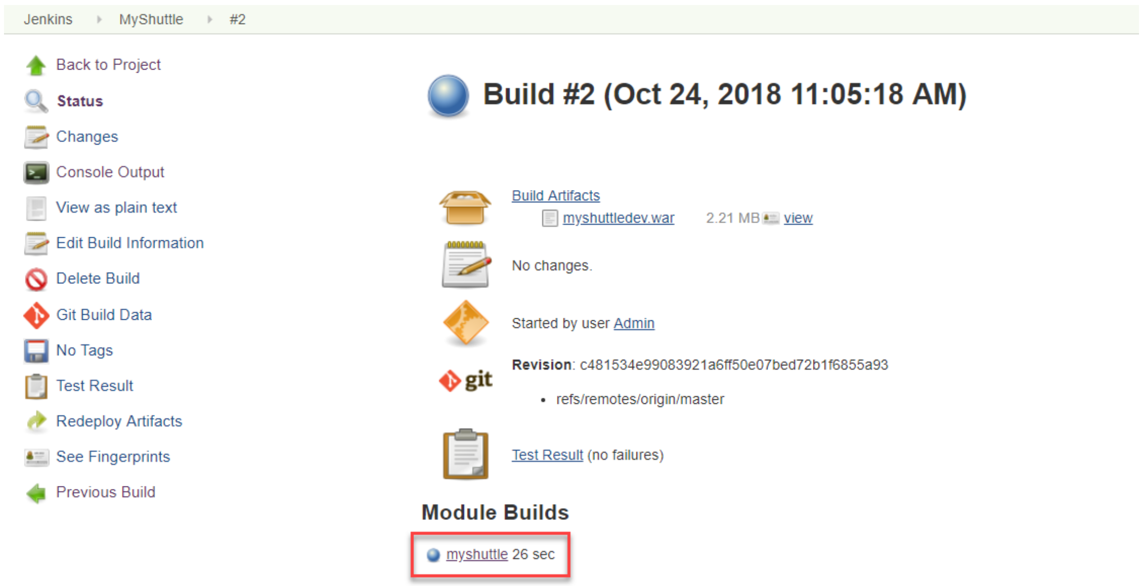


Maven project MyShuttle

Build History: #1 Nov 6, 2017 11:20 PM

Page generated: Nov 6, 2017 11:20:30 PM UTC REST API Jenkins ver. 2.73.2

10. To view the build details and the list of build artifacts, select the build number displayed in the **Module Builds** section.


Jenkins › MyShuttle › #2

- Back to Project
- Status
- Changes
- Console Output
- View as plain text
- Edit Build Information
- Delete Build
- Git Build Data
- No Tags
- Test Result
- Redeploy Artifacts
- See Fingerprints
- Previous Build

Build #2 (Oct 24, 2018 11:05:18 AM)

Build Artifacts
myshuttledev.war 2.21 MB view

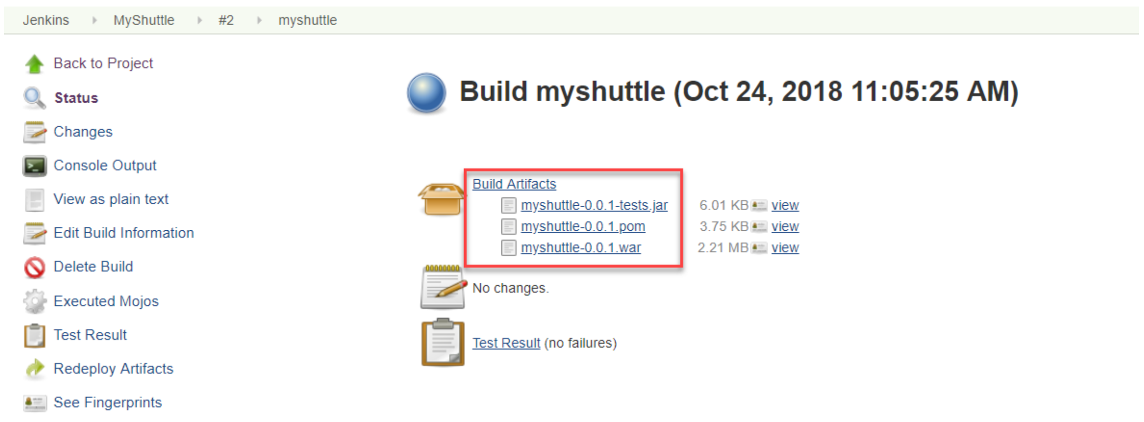
No changes.

Started by user Admin

Revision: c481534e99083921a6ff50e07bed72b1f6855a93
- refs/remotes/origin/master

Test Result (no failures)

Module Builds
myshuttle 26 sec


Jenkins › MyShuttle › #2 › myshuttle

- Back to Project
- Status
- Changes
- Console Output
- View as plain text
- Edit Build Information
- Delete Build
- Executed Mojos
- Test Result
- Redeploy Artifacts
- See Fingerprints

Build myshuttle (Oct 24, 2018 11:05:25 AM)

Build Artifacts
myshuttle-0.0.1-tests.jar 6.01 KB view
myshuttle-0.0.1.pom 3.75 KB view
myshuttle-0.0.1.war 2.21 MB view

No changes.

Test Result (no failures)

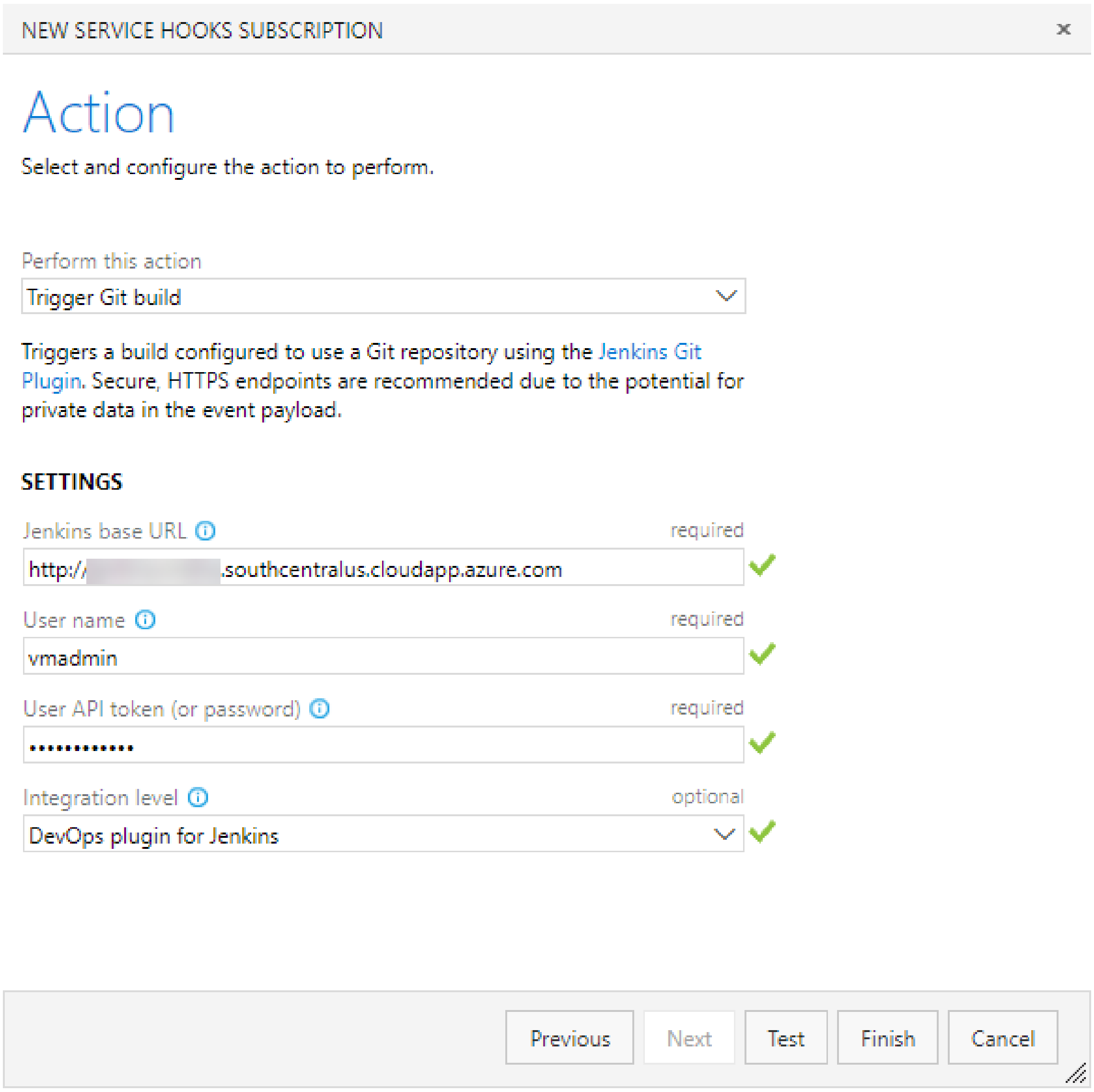
11. Select the **Test Result** option to view the results of the unit tests that were included in the build definition.

Next, we will explore the two different options available to trigger the Jenkins CI job when a code is pushed to Azure Repos.

Approach 1: Triggering the CI via a service hook in Azure DevOps

In this approach, a service hook will be configured in Azure DevOps to trigger a Jenkins build upon a code commit. Service hooks enable you to perform tasks on other services when events happen in your Azure DevOps Services projects.

1. To configure the service hook, navigate to the Azure DevOps project settings page and select **Service hooks** under **General**. Select **+ Create subscription**.
2. In the *New Service Hook Subscriptions* screen, select the **Jenkins** option and then click the **Next** button. Jenkins service supports three events - **Build completed**, **Code Pushed** and **Pull request merged**. We are only interested in the code push event - so, select **Code pushed** for the **Trigger on this type of event** field. Select the **MyShuttle** repository and then click **Next**


NEW SERVICE HOOKS SUBSCRIPTION

Action

Select and configure the action to perform.

Perform this action: Trigger Git build

Triggers a build configured to use a Git repository using the Jenkins Git Plugin. Secure, HTTPS endpoints are recommended due to the potential for private data in the event payload.

SETTINGS

Jenkins base URL (required): http://[redacted].southcentralus.cloudapp.azure.com

User name (required): vmadmin

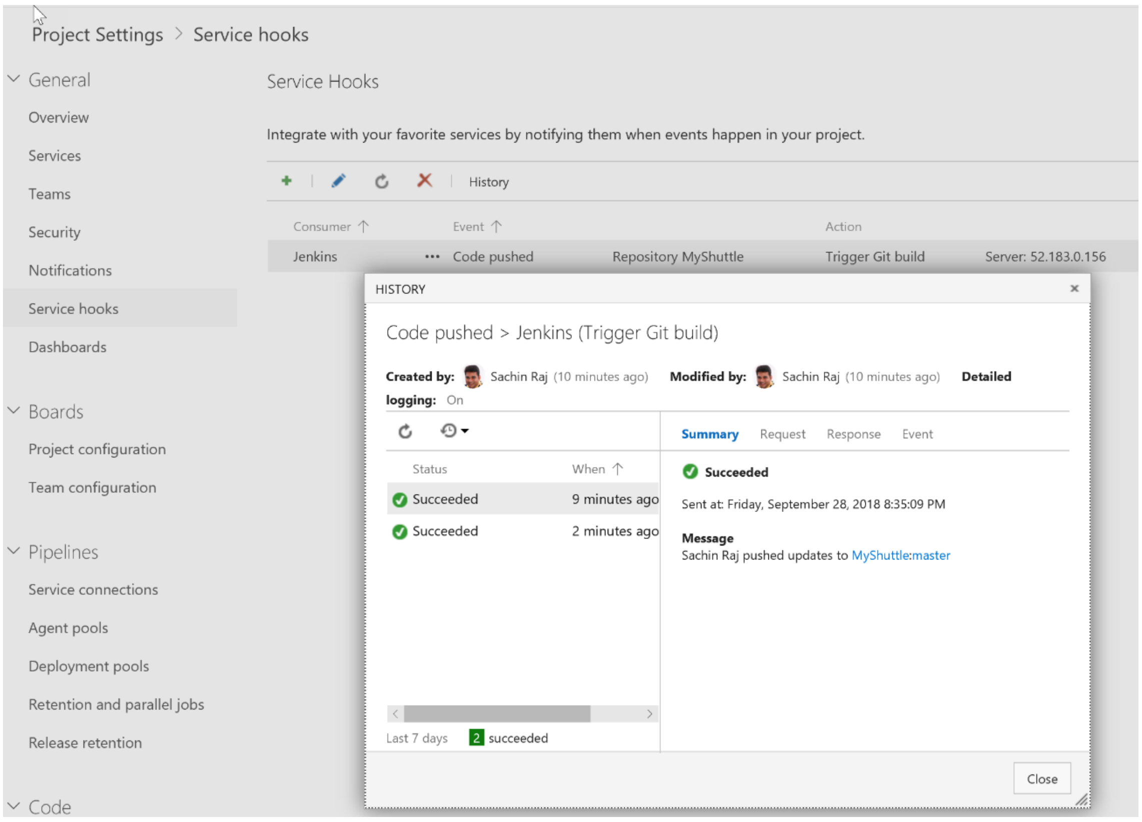
User API token (or password) (required): ••••••••••••

Integration level (optional): DevOps plugin for Jenkins

3. Provide the following details in the **Select and configure the action to perform** screen

    a. Select the **Trigger Git build** option
    b. Provide the **Jenkins base URL** in `http://{ip address or the host name}` format
    c. Provide the **User name** and **Password** to trigger the build. Note the username and password is the credentials of the Jenkins administrator user that you configured earlier

4. Click the **Test** button to validate the configuration and then click **Finish**. This will set the trigger to initiate the Jenkins CI build whenever a source code change is committed on the repository.
5. Try making a commit to the code - `src/main/webapp/index.jsp` would be a good candidate. This should trigger the MyShuttle build on Jenkins. You can confirm it by checking the history tab of the Jenkins services hook.



Integrate with your favorite services by notifying them when events happen in your project.

Consumer	Event		Action	
Jenkins	Code pushed	Repository MyShuttle	Trigger Git build	Server: 52.183.0.156

HISTORY

Code pushed > Jenkins (Trigger Git build)

Created by: Sachin Raj (10 minutes ago) **Modified by:** Sachin Raj (10 minutes ago) **Detailed logging:** On

Status	When
Succeeded	9 minutes ago
Succeeded	2 minutes ago

Last 7 days 2 succeeded

Summary Request Response Event

Succeeded

Sent at: Friday, September 28, 2018 8:35:09 PM

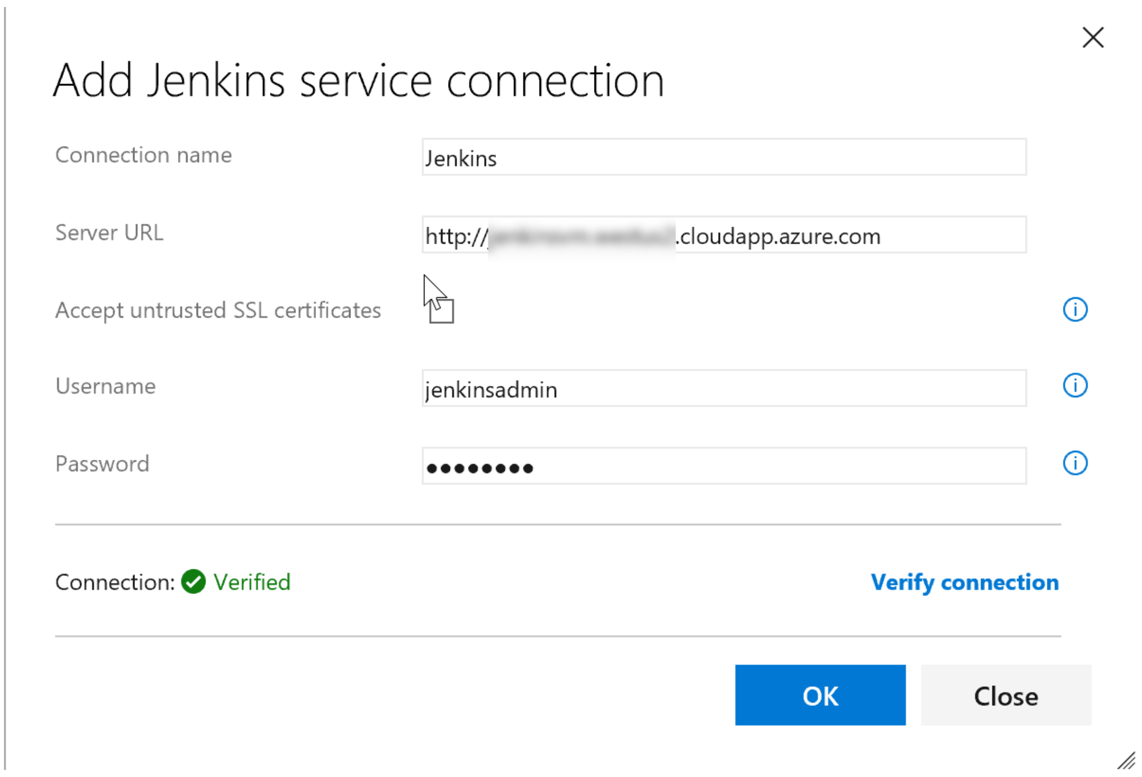
Message
Sachin Raj pushed updates to MyShuttle:master

Approach 2: Wrapping Jenkins Job within Azure Pipelines

In this approach, Jenkins CI job will be nested within an Azure CI pipeline. The key benefit of this approach is you can have end-to-end traceability from work items to source code to build and release pipelines.

To begin, an endpoint to the Jenkins Server for communication with Azure DevOps will be configured.

1. Go to your project settings. Select **Pipelines** and **Service connections**, click **New service connection** and choose **Jenkins** from the dropdown.
2. Provide a connection name, Jenkins server URL in the format `http://[server IP address or DNS name]` and Jenkins user name with password. Select **Verify Connection** and validate the configuration. If it successful, then select **Ok**.



Add Jenkins service connection

Connection name: Jenkins

Server URL: http://[redacted].cloudapp.azure.com

Accept untrusted SSL certificates: ☐

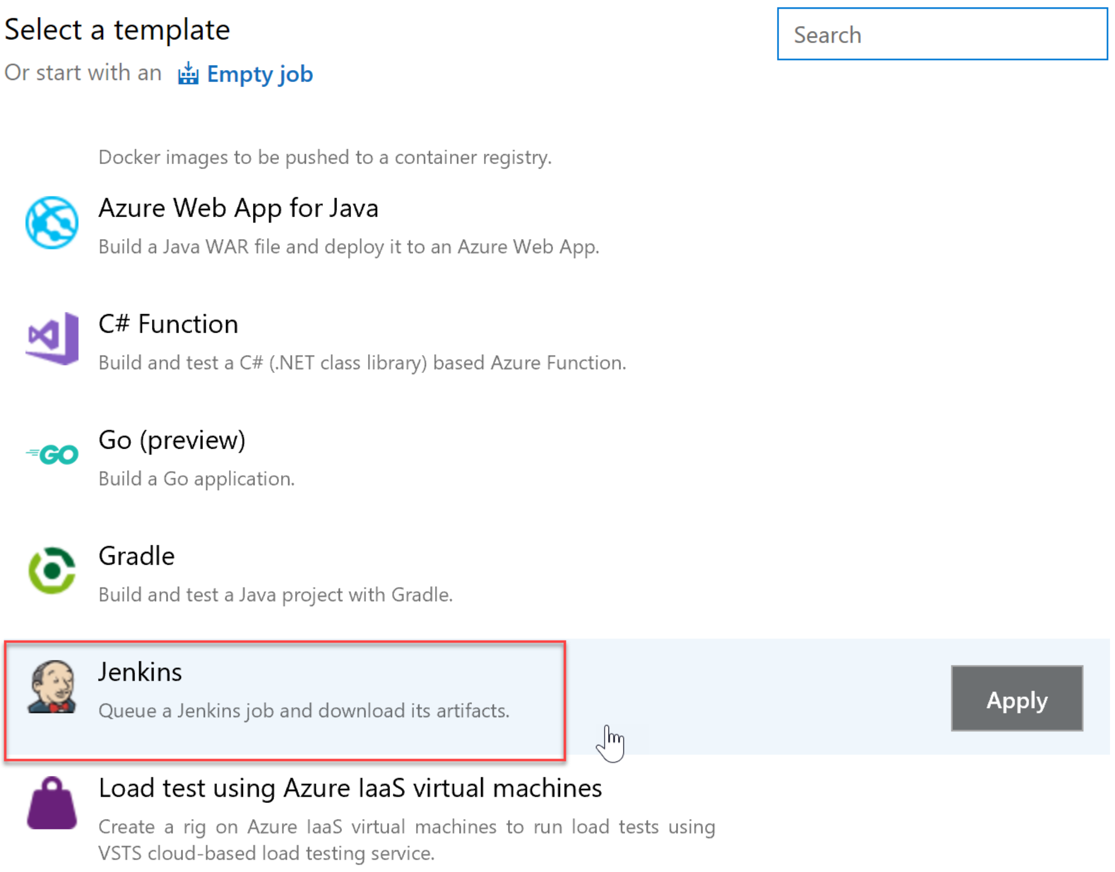
Username: jenkinsadmin

Password: ●●●●●●●●●

Connection: ✓ Verified

The next step would be to configure the build pipeline.

3. Go to **Azure Pipelines** and **Builds**, Click **+New** and select **New build pipeline** to create a new build definition.
4. At the time of writing this lab, Azure Pipelines did not support Jenkins in YAML. Select **Use the Visual Designer** to create a pipeline without a YAML.
5. Select **Myshuttle** project, repository and click *Continue*.
6. Scroll down and select the standard **Jenkins** template Click Apply.


Select a template

Or start with an Empty job

Search

Docker images to be pushed to a container registry.

Azure Web App for Java
Build a Java WAR file and deploy it to an Azure Web App.

C# Function
Build and test a C# (.NET class library) based Azure Function.

Go (preview)
Build a Go application.

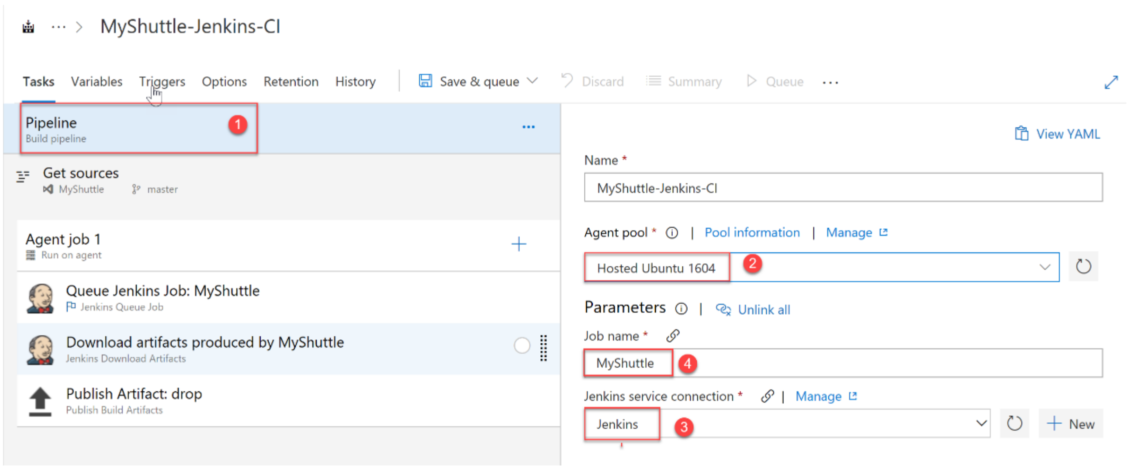
Gradle
Build and test a Java project with Gradle.

Jenkins
Queue a Jenkins job and download its artifacts.

Apply

Load test using Azure IaaS virtual machines
Create a rig on Azure IaaS virtual machines to run load tests using VSTS cloud-based load testing service.

7. Select **Hosted VS2017** for the Agent Queue, provide **MyShuttle** as the Job name (name of the build definition that was created in Jenkins) and then select the Jenkins service endpoint created earlier.


MyShuttle-Jenkins-CI

Tasks | Variables | Triggers | Options | Retention | History | Save & queue | Discard | Summary | Queue

Pipeline — Build pipeline (1)

Get sources — MyShuttle — master

Agent job 1 — Run on agent

Queue Jenkins Job: MyShuttle — Jenkins Queue Job

Download artifacts produced by MyShuttle — Jenkins Download Artifacts

Publish Artifact: drop — Publish Build Artifacts

View YAML

Name *: MyShuttle-Jenkins-CI

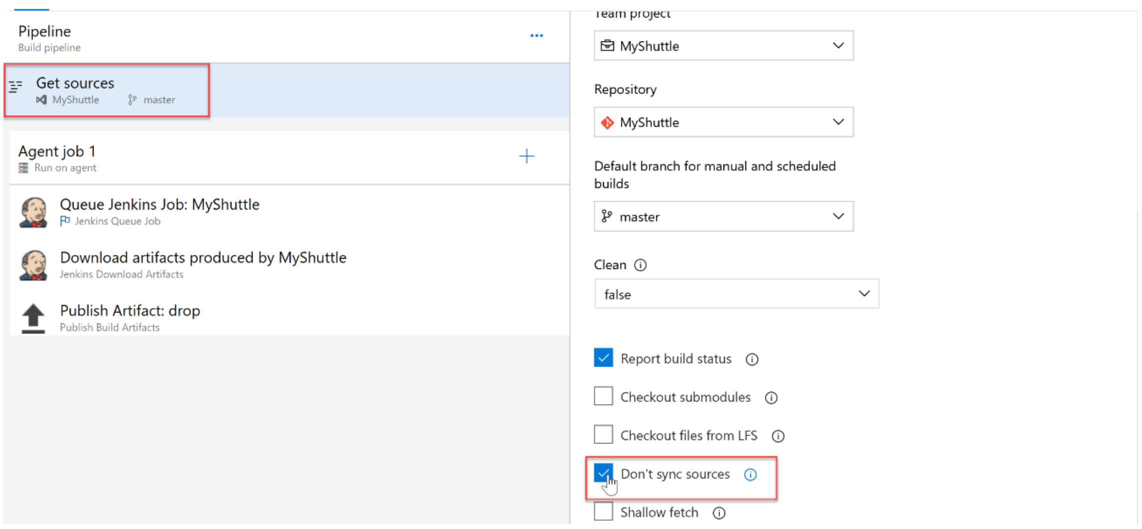
Agent pool * | Pool information | Manage: Hosted Ubuntu 1604 (2)

Parameters | Unlink all

Job name *: MyShuttle (4)

Jenkins service connection * | Manage: Jenkins (3) | New

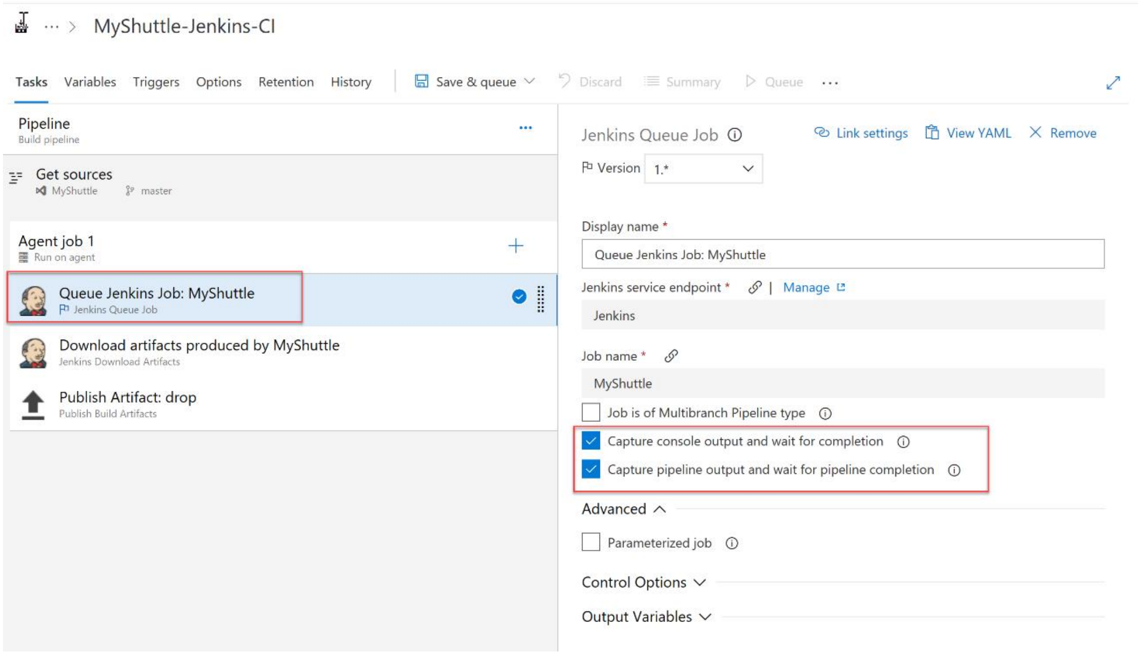
8. Next, select the **Get Sources** step. Since Jenkins is being used for the build, there is no need to download the source code to the build agent. To skip syncing with the agent, select **Don’t sync sources** option.


Pipeline
Build pipeline
Get sources
MyShuttle master
Agent job 1
Run on agent
Queue Jenkins Job: MyShuttle
Jenkins Queue Job
Download artifacts produced by MyShuttle
Jenkins Download Artifacts
Publish Artifact: drop
Publish Build Artifacts

Team project: MyShuttle
Repository: MyShuttle
Default branch for manual and scheduled builds: master
Clean: false
[✓] Report build status
[ ] Checkout submodules
[ ] Checkout files from LFS
[✓] Don't sync sources
[ ] Shallow fetch

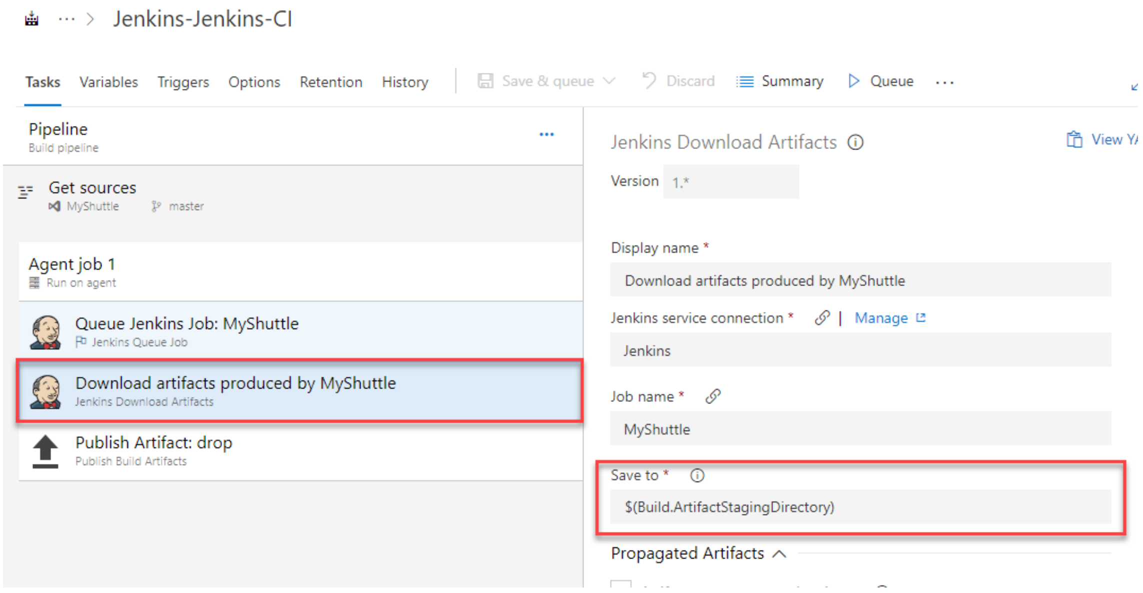
9. Next, select the **Queue Jenkins Job** step. This task queues the job on the Jenkins server. Make sure that the services endpoint and the job name are correct. The **Capture console output** and the **Capture pipeline output** options available at this step will be selected.

   The **Capture console output and wait for completion** option, when selected, will capture the output of the Jenkins build console when the Azure build pipeline runs. The build will wait until the Jenkins Job is completed. The **Capture pipeline output and wait for pipeline completion** option is very similar but applies to Jenkins pipelines (a build that has more than one job nested together).


... > MyShuttle-Jenkins-CI
Tasks Variables Triggers Options Retention History Save & queue Discard Summary Queue ...
Link settings View YAML Remove
Pipeline
Build pipeline
Get sources
MyShuttle master
Agent job 1
Run on agent
Queue Jenkins Job: MyShuttle
Jenkins Queue Job
Download artifacts produced by MyShuttle
Jenkins Download Artifacts
Publish Artifact: drop
Publish Build Artifacts

Jenkins Queue Job
Version: 1.*
Display name *: Queue Jenkins Job: MyShuttle
Jenkins service endpoint *: Jenkins
Job name *: MyShuttle
[ ] Job is of Multibranch Pipeline type
[✓] Capture console output and wait for completion
[✓] Capture pipeline output and wait for pipeline completion
Advanced
[ ] Parameterized job
Control Options
Output Variables

10. The **Jenkins Download Artifacts** task will download the build artifacts from the Jenkins job to the staging directory.



Jenkins-Jenkins-CI

Pipeline
Build pipeline

Get sources
MyShuttle master

Agent job 1
Run on agent

Queue Jenkins Job: MyShuttle
Jenkins Queue Job

Download artifacts produced by MyShuttle
Jenkins Download Artifacts

Publish Artifact: drop
Publish Build Artifacts

Jenkins Download Artifacts

Version: 1.*

Display name *: Download artifacts produced by MyShuttle

Jenkins service connection *: Jenkins

Job name *: MyShuttle

Save to *: $(Build.ArtifactStagingDirectory)

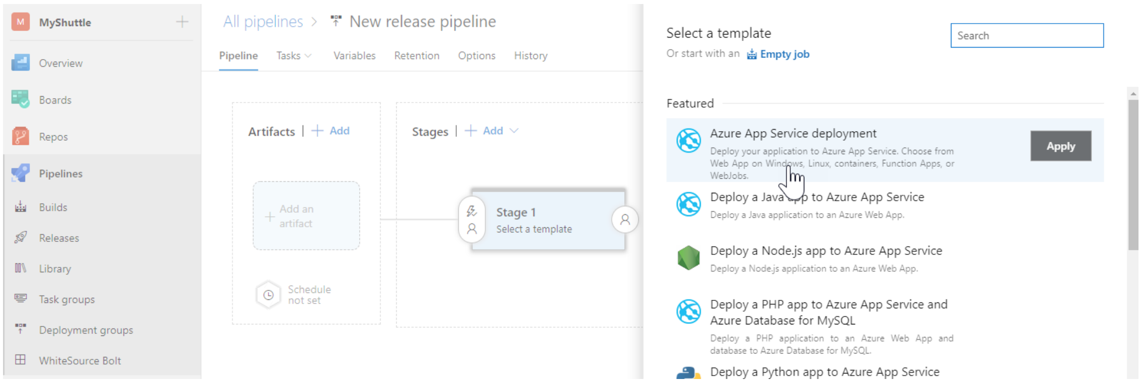
Propagated Artifacts

11. The **Publish Artifact drop** will publish to Azure Pipelines.
12. Click **Save & queue** button to save and initiate a new build.

Linking the build artifact for deployment in a CD pipeline

Next, you will configure an Azure CD pipeline to fetch and deploy the artifacts produced by the build. Since the deployment is being done to Azure, an endpoint to Azure will be configured. An endpoint to Jenkins server will also be configured, if not configured earlier.

1. After the endpoint creation, go to the **Releases** tab in **Azure Pipelines**. Open the + drop-down in the list of release pipelines, and choose **Create release pipeline**.
2. Select the **Azure App Service Deployment** template.



Select a template
Or start with an Empty job

Featured

Azure App Service deployment
Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.

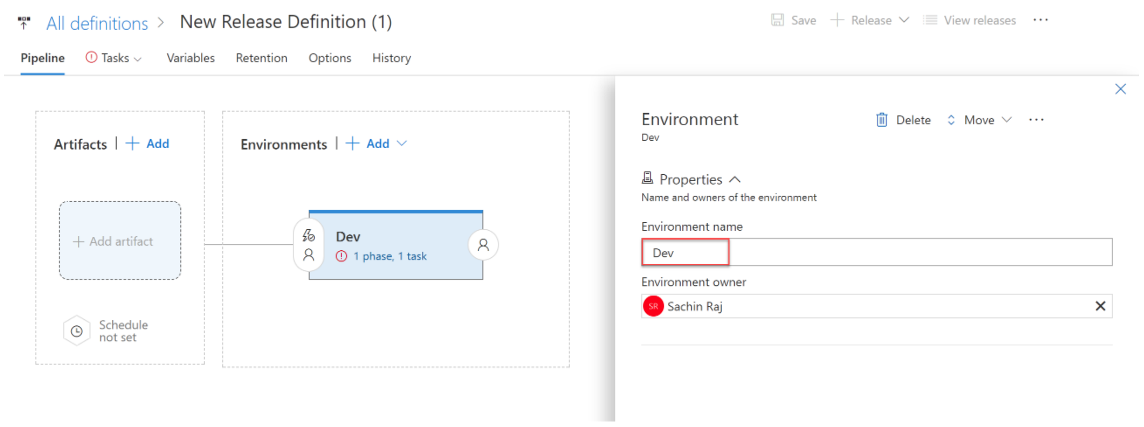
Deploy a Java app to Azure App Service
Deploy a Java application to an Azure Web App.

Deploy a Node.js app to Azure App Service
Deploy a Node.js application to an Azure Web App.

Deploy a PHP app to Azure App Service and Azure Database for MySQL
Deploy a PHP application to an Azure Web App and database to Azure Database for MySQL.

Deploy a Python app to Azure App Service

3. The default environment for deployment will be named as **Dev**

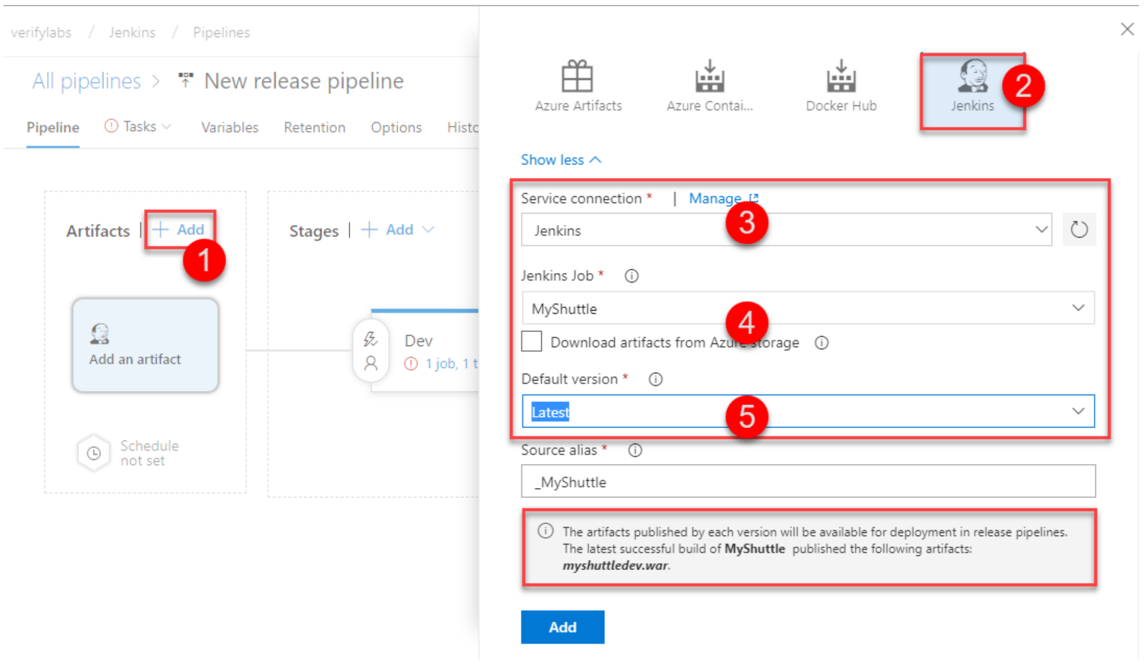


Environment name: Dev

Environment owner: Sachin Raj

4. In the **Artifacts** section in the **Pipeline** tab, choose the **+ Add** link to select your build artifact.
   a. If you have used the first approach, select **Jenkins** as the *Source type*, select the Jenkins endpoint configured earlier and provide **MyShuttle** for the *Source(Job)*, choose the **Default version** as *Latest*. The Source(Job) should map to the project name configured in Jenkins.

      If the Jenkins server and the source location is configured correctly, once the publishing of the artifacts is completed, a message with the output file name **myshuttledev.war** will be displayed.



Service connection * | Manage: Jenkins

Jenkins Job *: MyShuttle

Download artifacts from Azure storage

Default version *: Latest

Source alias *: _MyShuttle

The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of **MyShuttle** published the following artifacts: ***myshuttledev.war***.

   b. Otherwise, point this to the Azure CI build pipeline from which the Jenkins CI is executed.
5. Now, the artifact is linked for deployment. Please refer the Deploying a MySQL Backed Tomcat app on Azure Web App lab for deploying the WAR file to Azure App Service.